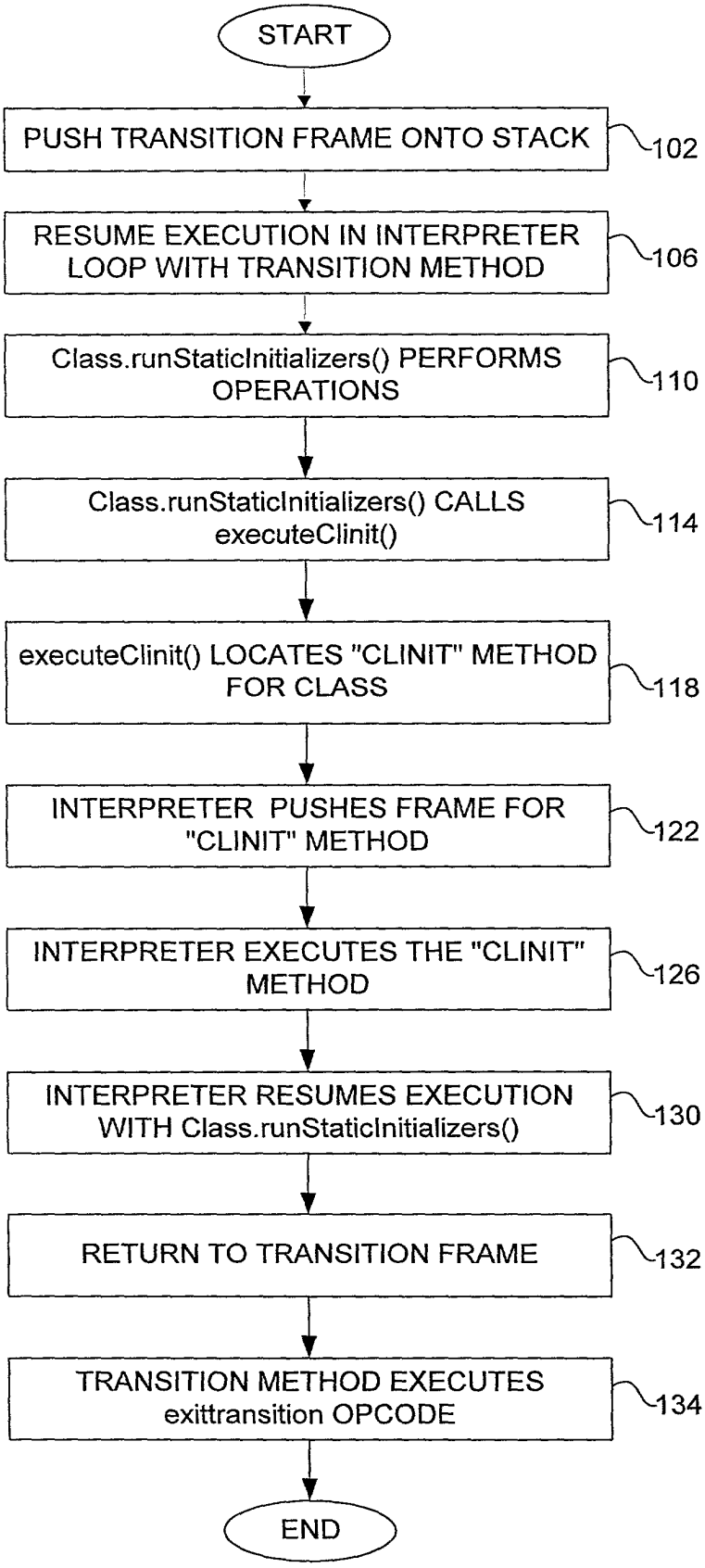
START
PUSH TRANSITION FRAME ONTO STACK
102
RESUME EXECUTION IN INTERPRETER LOOP WITH TRANSITION METHOD
106
Class.runStaticInitializers() PERFORMS OPERATIONS
110
Class.runStaticInitializers() CALLS executeClinit()
114
executeClinit() LOCATES "CLINIT" METHOD FOR CLASS
118
INTERPRETER PUSHES FRAME FOR "CLINIT" METHOD
122
INTERPRETER EXECUTES THE "CLINIT" METHOD
126
INTERPRETER RESUMES EXECUTION WITH Class.runStaticInitializers()
130
RETURN TO TRANSITION FRAME
132
TRANSITION METHOD EXECUTES exittransition OPCODE
134
END

Figure 1

```
private void runStaticInitializers() throws Throwable {
    CVM.disableRemoteExceptions();
    try {

      /* Step 1 - lock the class. */
      synchronized (this) {

          /* Step 2 - if someone is currenly initializing this class
           * and it's not our thread, then wait until the class is
           * done initializing. */
          while (checkInitializingFlag(false)) {
            try {
                wait(0);
            } catch (InterruptedException e) {
                /* This can't happen since async exceptions are
                 * disabled during static intializers.
                 */
            }
          }

          /* Step 3 and 4 - return success if this thread is
           * initializing the class or if the class is already
           * initialized. */
          if (checkInitializingFlag(true) || checkInitializedFlag()) {
            return;
          }

          /* Step 5 - fail if class is already in an erroneous state. */
          if (checkErrorFlag()) {
            throw new NoClassDefFoundError(getName());
          }

          /* Step 6 - mark the class as "initializing" and unlock it. */
          setInitializingFlag();
      }

      /* Step 7 - run the superclass static initialzer. */
      Class superClass = getSuperclass();
      if (superClass != null && !superClass.checkInitializedFlag()) {
          try {
            superClass.runStaticInitializers();
          } catch (Throwable e) {
            synchronized (this) {
                clearInitializingFlag();
                setErrorFlag();
                notifyAll();
            }
            throw e;
          }
      }

      /* Step 8 - run the static intializer. */
      Throwable clinitError = null;
      try {
          CVM.executeClinit(this);
      } catch (Throwable e) {
```

Figure 2

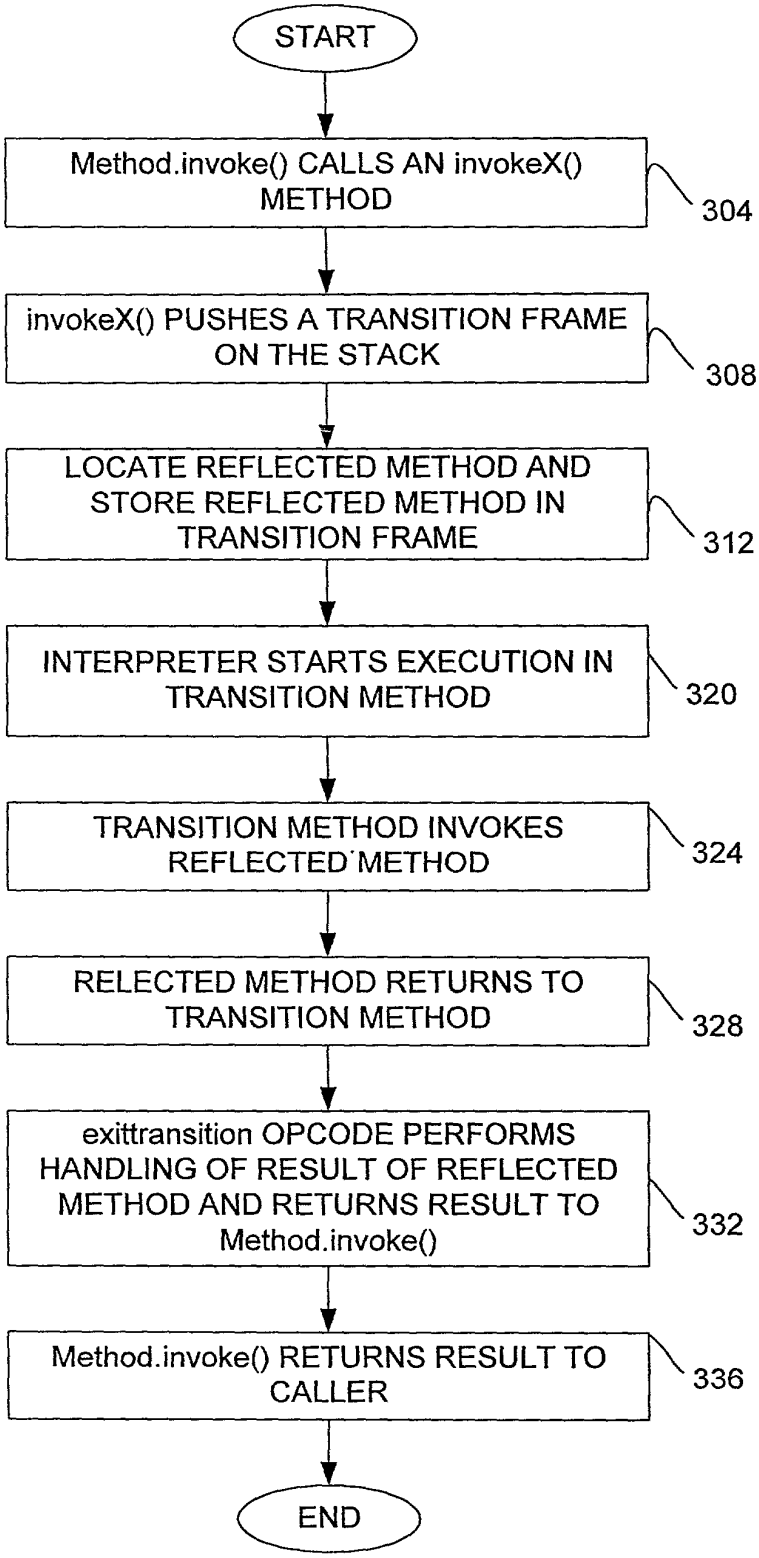
START
Method.invoke() CALLS AN invokeX() METHOD
304
invokeX() PUSHES A TRANSITION FRAME ON THE STACK
308
LOCATE REFLECTED METHOD AND STORE REFLECTED METHOD IN TRANSITION FRAME
312
INTERPRETER STARTS EXECUTION IN TRANSITION METHOD
320
TRANSITION METHOD INVOKES REFLECTED METHOD
324
RELECTED METHOD RETURNS TO TRANSITION METHOD
328
exittransition OPCODE PERFORMS HANDLING OF RESULT OF REFLECTED METHOD AND RETURNS RESULT TO Method.invoke()
332
Method.invoke() RETURNS RESULT TO CALLER
336
END

Figure 3

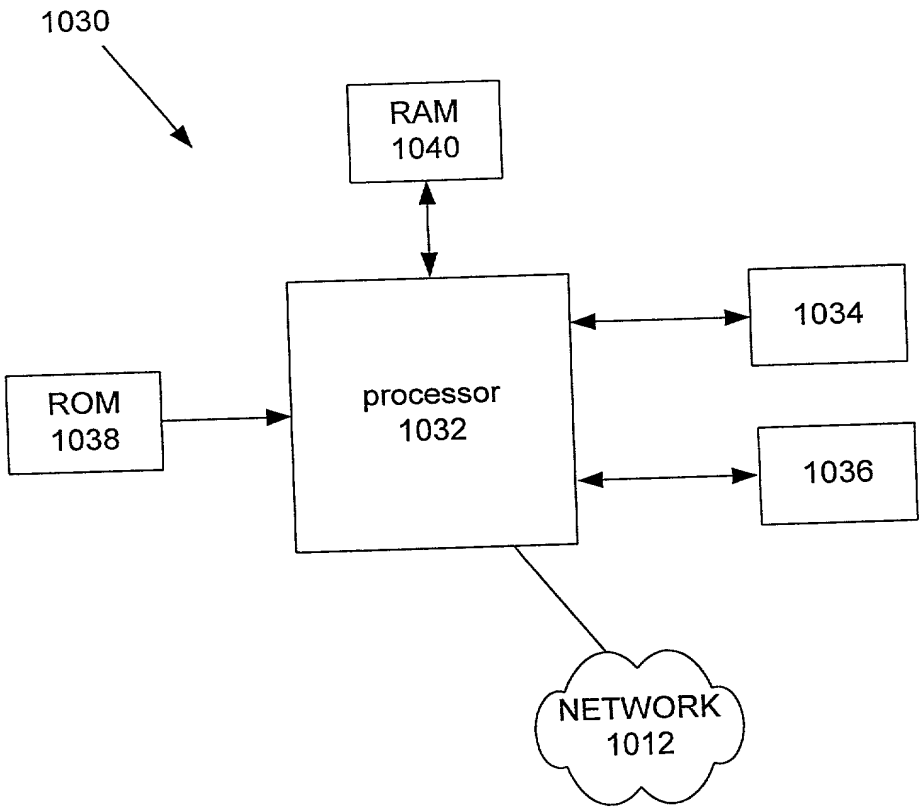
1030
RAM
1040
processor
1032
ROM
1038
1034
1036
NETWORK
1012

Figure 4

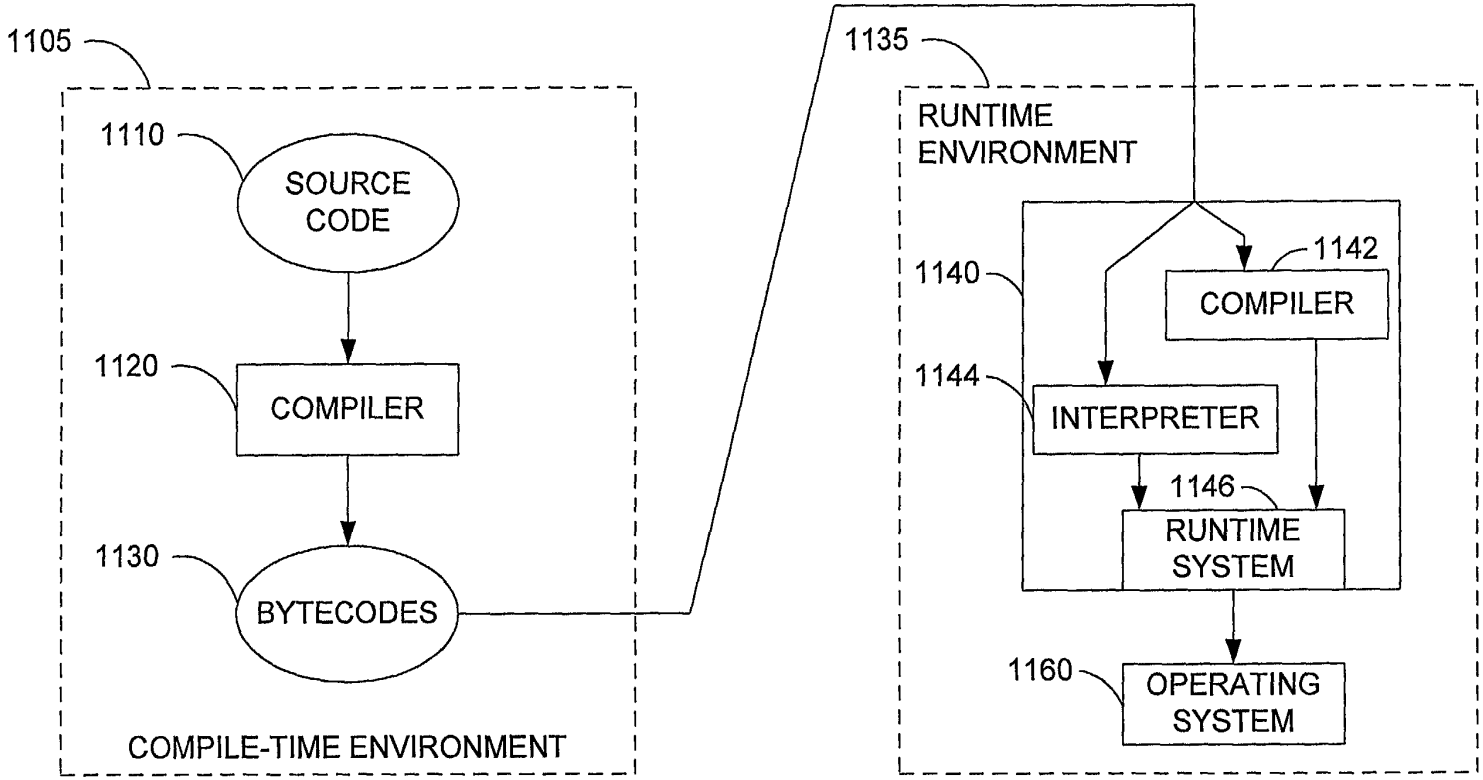
1105
1110
SOURCE CODE
1120
COMPILER
1130
BYTECODES
COMPILE-TIME ENVIRONMENT
1135
RUNTIME ENVIRONMENT
1140
1142
COMPILER
1144
INTERPRETER
1146
RUNTIME SYSTEM
1160
OPERATING SYSTEM

Figure 5